NRL Memorandum Report 4702
December 8, 1981

A-7E SOFTWARE MODULE GUIDE

by

Kathryn Heninger Britton

and

David Lorge Parnas

Revisions
by
Paul C. Clements
Lisa A. Kurowski

I. INTRODUCTION

PURPOSE

The A-7E Module Guide describes the module structure of the A-7E flight software produced by the Naval Research Laboratory. It provides an orientation for software engineers who are new to the A-7E system, explains the principles used to design the structure, and shows how responsibilities are allocated among the major modules.

This guide is intended to lead a reader to the module that deals with a particular aspect of the system. It states the criteria used to assign a particular responsibility to a module and arranges the modules in such a way that a reader can find the information relevant to his purpose without searching through unrelated documentation.

The module guide should be read before any other internal design documentation for the NRL A-7E software, because the guide defines the scope and contents of the individual design documents.

This guide describes and prescribes the module structure. Changes in the structure will be promulgated as changes to this document. Changes are not official until they appear in that form. This guide is a rationalization of the structure, not a description of the design process that led to it.

PREREQUISITE KNOWLEDGE

Readers are assumed to be familiar with the terminology and organization of [1], which will be referred to as ''the requirements document''. They should have a general idea of the functions performed by the A-7E flight software, and be familiar with the types of hardware devices that are connected to such computers. These are described in the requirements document.

ORGANIZATION

Section II gives the background for the design. It states 1) the goals that motivated the module design decisions presented in this document; 2) the basic principles on which the design is based; and 3) the relationship between the module structure and two other structures of the NRL A-7E software.

Section III, the main body of the document, presents a hierarchical decomposition of the software into top-level, second-level, and third-level modules. The modules at each level are components of modules of the next higher level.

Terms, such as ''module'', that are used with a special meaning in NRL's A-7E design, are defined in the glossary. Readers who are not familiar with the NRL A-7E project's terminology should study the glossary before reading further. An overview of the methodology is also given in [15, 16].

II. BACKGROUND

THE A-7E SOFTWARE STRUCTURES

A structural description of a software system shows the program's decomposition into parts and the relations between those parts. A-7E programmers must be concerned with three structures: (1) the **module** structure, (2) the **uses** structure, and (3) the **process** structure. This section contrasts these structures.

(1) Each module consists of a group of closely related programs. The **module structure** is the decomposition of the program into modules and the assumptions that the team responsible for each module is allowed to make about the other modules.

(2) In the **uses structure** the components are programs, i.e., not modules but parts of modules; the relation is ''requires the correct presence of''. The uses structure determines the executable subsets of the software [7]. Guidelines for the design of the A-7E uses structure are given in [7].

(3) The **process structure** is a decomposition of the run-time activities of the system into units known as processes. Processes are not programs; there is no simple relation between modules and processes. The implementation of some modules may include one or more processes, and any process may invoke programs in several modules.

The rest of this document describes the module structure.

GOALS OF THE A-7E MODULE STRUCTURE

The overall goal of the decomposition into modules is reduction of software cost by allowing modules to be designed, implemented, and revised independently. Specific goals of the module decomposition are:

(1) each module's structure should be simple enough that it can be understood fully;

(2) it should be possible to change the implementation of one module without knowledge of the implementation of other modules and without affecting the behavior of other modules;

(3) the ease of making a change in the design should bear a reasonable relationship to the likelihood of the change being needed; it should be possible to make likely changes without changing any module interfaces; less likely changes may involve interface changes, but only for modules that are small and not widely used. Only very unlikely changes should require changes in the interfaces of widely used modules. There should be few widely used interfaces;

(4) it should be possible to make a major software change as a set of independent changes to individual modules, i.e., except for interface changes, programmers changing the individual modules should not need to communicate. If the interfaces of the modules are not revised, it should be possible to run and test any combination of old and new module versions.

As a consequence of the goals above, the A-7E software is composed of many small modules. They have been organized into a tree-structured hierarchy; each nonterminal node in the tree represents a module that is **composed of** the modules represented by its descendents. The hierarchy is intended to achieve the following additional goals:

(5) A software engineer should be able to understand the responsibility of a module without understanding the module's internal design.

(6) A reader with a well defined concern should easily be able to identify the relevant modules without studying irrelevant modules. This implies that the reader be able to distinguish relevant modules from irrelevant modules without looking at their internal structure.

DESIGN PRINCIPLE

The A-7E module structure is based on the decomposition criteria known as information hiding [2]. According to this principle, system details that are likely to change independently should be the secrets of separate modules; the only assumptions that should appear in the interfaces between modules are those that are considered unlikely to change. Every data structure is private to one module; it may be directly accessed by one or more programs within the module but not by programs outside the module. Any other program that requires information stored in a module's data structures must obtain it by calling module programs.

Applying this principle is not always easy. It is an attempt to minimize the expected cost of software and requires that the designer estimate the likelihood of changes. Such estimates are based on past experience, and may require knowledge of the application area, as well as an understanding of hardware and software technology.

In a few cases information that is likely to change must be communicated between modules. To reduce the cost of software changes, use of some modules or portions of a module interface, may be restricted. Restricted interfaces are indicated by ''(R)'' in the documentation. Often the existence of certain smaller modules is itself a secret of a larger module. In a few cases, we have mentioned such modules in this document in order to clarify where certain functions are performed. Those modules are referred to as hidden modules and indicated by ''(H)'' in the documentation.

MODULE DESCRIPTION

Three ways to describe a module structure based on information-hiding are: (1) by the **roles** played by the individual modules in the overall system operation; (2) by the **secrets** associated with each module; and (3) by the **facilities** provided by each module. This document describes the module structure by characterizing each module's secrets. Where useful, we also include a brief description of the role of the module. The description of facilities is relegated to the module specifications (e.g. [3,4,5,6,9,10,12,13,14]).

For some modules we find it useful to distinguish between a **primary secret**, which is hidden information that was specified to the software designer, and a **secondary secret**, which refers to implementation decisions made by the designer when implementing the module designed to hide the primary secret.

Although we have attempted to make the decomposition rules as precise as possible, the possibility of future changes in technology makes some of the boundaries fuzzy. Some sections point out fuzzy areas and discuss additional information that must be used to resolve ambiguities.

MODULE INITIALIZATION

Every module in the A-7E software can contain variables that must be given initial values when the computer is turned on. Each module contains a program for initialization that will be called when power up occurs. There will be a main initialization program that is invoked at power up. It will invoke the initialization programs for selected second-level modules. The initialization program for a module will call the initialization programs for each of its submodules. These programs will be executed, sequentially, before any of the parallel processes begin to execute.

The initialization program for a module at a leaf of the module tree is part of that module. The initialization program for a higher-level module M is contained in a submodule of M, called the **Initialization Module** of M. These initialization modules will not be described further in this document. The initialization program for the system is contained in the module defined in Section C:3.5.1 of this document.

TREATMENT OF UNDESIRED EVENTS

Development versions of all modules will check for and report undesired events (UEs). Each module interface description contains a list of possible UEs. It should include hardware errors, software errors and errors caused by the using program. During development, the implementation of a module will

check for the specified UEs. The user of a module may supply a program to be invoked when the UE is detected. However, much of the UE detection and correction must be removed from the production version of the system because of space limitations.

The remainder of this report provides a top-down overview of the module structure.

III. A-7E MODULE STRUCTURE

A: TOP-LEVEL DECOMPOSITION

The software system consists of the three modules described below.

A:1 HARDWARE-HIDING MODULE

The Hardware-Hiding Module includes the programs that need to be changed if any part of the hardware is replaced by a new unit with a different hardware/software interface but with the same general capabilities. This module implements ‘‘virtual hardware’’ or an abstract device that is used by the rest of the software. The primary secrets of this module are the hardware/software interfaces described in chapters 1 and 2 of the requirements document. The secondary secrets of this module are the data structures and algorithms used to implement the virtual hardware.

A:2 BEHAVIOR-HIDING MODULE

The Behavior-Hiding Module includes programs that need to be changed if there are changes in the sections of the requirements document that describe the required behavior (chapters 3 and 4). The content of those sections is the primary secret of this module. These programs determine the values to be sent to the virtual output devices provided by the Hardware-Hiding Module.

A:3 SOFTWARE DECISION MODULE

The Software Decision Module hides software design decisions that are based upon mathematical theorems, physical facts, and programming considerations such as algorithmic efficiency and accuracy. The secrets of this module are **not** described in the requirements document. This module differs from the other modules in that both the secrets and the interfaces are determined by software designers. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

Notes on the top-level decomposition:

Fuzziness in the above classifications is unavoidable for the following reasons:

(1) The line between requirements definition and software design has been determined in part by decisions made when the requirements documents are written; for example, weapon trajectory models may be chosen by system analysts and specified in the requirements document, or they may be left to the discretion of the software designers.

(2) The line between hardware characteristics and software design may vary. Hardware can be built to perform some of the services currently performed by the software; consequently, certain modules can be viewed either as modules that hide hardware characteristics or as modules that hide software design decisions.

(3) Changes in the hardware or in the behavior of the system or its users may make a software design decision less appropriate.

(4) All software modules include software design decisions; changes in any module may be motivated by efficiency or accuracy considerations.

To reduce the fuzziness, we have based our decomposition on the current requirements document. In particular,

(1) The line between requirements and software design is defined by our requirements document. When the requirements document specifies an algorithm, we do not consider the design of the algorithm to be a software design decision. If the requirements document only states requirements that the algorithm must meet, we consider the program that implements that algorithm to be part of a Software Decision Module.

(2) The line between hardware characteristics and software design is based on estimates of the likelihood of future changes. For example, if it is reasonably likely that future hardware will implement a particular facility, the software module that implements that facility is classified as a hardware-hiding module; otherwise, the module is considered a software design module. In most cases we have taken a conservative stance; the design is based on the assumption that drastic changes are less likely than evolutionary changes.

(3) A module is included in the Software Decision Module only if it would remain correct, albeit less efficient, when there are changes in the requirements document.

(4) A module will be included in the Software Decision Module only if its secrets do not include information documented in the software requirements document.

B: SECOND-LEVEL DECOMPOSITION

B:1 HARDWARE-HIDING MODULE DECOMPOSITION:

The Hardware-Hiding Module comprises two modules.

B:1.1 EXTENDED COMPUTER MODULE

The Extended Computer Module hides those characteristics of the hardware/software interface of the avionics computer that we consider likely to change if the computer is modified or replaced.

Avionics computers differ greatly in their hardware/software interfaces and in the capabilities that are implemented directly in the hardware. Some avionics computers include a hardware approximation of real numbers, while others perform approximate real number operations by a programmed sequence of fixed-point operations. Some avionics systems include a single processor; some systems provide several processors. The Extended Computer provides an instruction set that can be implemented efficiently on most avionics computers. This instruction set includes the operations on application-independent data types, sequence control operations, and general I/O operations. The Extended Computer is a multi-processor but may also serve as a single processor.

The primary secrets of the Extended Computer are: the number of processors, the instruction set of the computer, the processor state transitions, the processor addressing restrictions, and the processor's self-test capabilities.

The structure of the Extended Computer Module is given in section C:1.1. Specifications for Extended Computer submodules are given in [4].

B:1.2 DEVICE INTERFACE MODULE

The Device Interface Module hides the peripheral device characteristics that are considered likely to change. Each device might be replaced by an improved device capable of accomplishing the same tasks. Replacement devices differ widely in their hardware/software interfaces. For example, all angle-of-attack sensors measure angle-of-attack, but they differ in input format, timing, and the amount of noise in the data.

The Device Interface Module provides virtual devices to be used by the rest of the software. The virtual devices do not necessarily correspond one to one to physical devices because all of the hardware providing a capability is not necessarily in one physical unit. Further, there are some capabilities of a physical unit that are likely to change independently of others; it is advantageous to hide characteristics that may change independently in different modules.

The primary secrets of the Device Interface Module are those characteristics of the present devices documented in the requirements document and not likely to be be shared by replacement devices.

The structure of the Device Interface Module is given in section C:1.2. Specifications for Device Interface submodules are given in [3].

Notes on the Hardware-Hiding Module Decomposition

Our distinction between computer and device is based on the current hardware and is the one made in the requirements document. Information that applies to more than one device is considered a secret of the Extended Computer; information that is only relevant to one device is a secret of a Device Interface Module. For example, there is an analog to digital converter that is used for communicating with several devices; it is hidden by the Extended Computer although it could be viewed as an external device. As another example, there are special outputs for testing the I/O channels; they are not associated with a single device. These too are hidden by the Extended Computer.

If all the hardware were replaced simultaneously, there might be a significant shift in responsibilities between computer and devices. In systems like the A-7E such changes are unusual; the replacement of individual devices or the replacement of the computer alone is more likely. Our design is based on the expectation that this pattern of replacement will continue to hold.

B:2 BEHAVIOR-HIDING MODULE DECOMPOSITION:

The Behavior-Hiding Module consists of two modules: a Function Driver (FD) Module supported by a Shared Services (SS) Module.

B:2.1 FUNCTION DRIVER MODULE

The Function Driver Module consists of a set of modules called Function Drivers; each Function Driver is the sole controller of a set of closely related outputs. The outputs are either part of a virtual device or provided by the Extended Computer for test purposes. The primary secrets of the Function Driver Module are the rules determining the values of these outputs.

The structure of the Function Driver Module is given in section C:2.1. Specifications for the Function Driver Module are found in [8].

B:2.2 SHARED SERVICES MODULE

Because all the Function Drivers control systems in the same aircraft, some aspects of the behavior are common to several Function Drivers. We expect that if there is a change in that aspect of the behavior, it will affect all of the functions that share it. Consequently we have identified a set of modules, each of which hides an aspect of the behavior that applies to two or more of the outputs.

The structure of the Shared Services Module is found in section C:2.2. Specifications for the Shared Services Module are found in [9].

Notes on the Behavior-Hiding Module structure

Because users of the documentation cannot be expected to know which aspects of a function's behavior are shared, the documentation for the Function Driver Modules will include a reference to the Shared Services Modules that it uses. A maintenance programmer should always begin his inquiry with the appropriate function driver. He will be directed to the Shared Services Modules when appropriate.

B:3 SOFTWARE DECISION MODULE DECOMPOSITION:

The Software Decision Module has been divided into (1) the Application Data Type Module, which hides the implementation of certain variables, (2) the Physical Models Module, which hides algorithms that simulate physical phenomena, (3) the Data Banker Module, which hides the data-updating policies, (4) the System Generation Module, which hides decisions that are postponed until system generation time, (5) the Software Utility Module, which hides algorithms that are used in several other modules.

B:3.1 APPLICATION DATA TYPE MODULE

The Application Data Type Module supplements the data types provided by the Extended Computer Module with data types that are useful for avionics applications and do not require a computer dependent implementation. These data types are implemented using the data types provided by the Extended Computer; variables of those types are used just as if the types were built into the Extended Computer.

The secrets of the Application Data Type Module are the data representation used in the variables and the programs used to implement operations on those variables. Units of measurements are part of the representation and are hidden. Where necessary, the modules provide conversion operators, which deliver or accept real values in specified units.

The structure of the Application Data Type Module is given in section C:3.1. Specifications for these modules may be found in [5].

B:3.2 DATA BANKER MODULE

Most data are produced by one module and consumed by another. In most cases, the consumers should receive a value as up-to-date as practical. The time at which a datum should be recalculated is

determined both by properties of its consumer (e.g., accuracy requirements) and by properties of its producer (e.g., cost of calculation, rate of change of value). The Data Banker Module acts as a ''middleman'' and determines when new values for these data are computed. The Data Banker obtains values from producer programs; consumer programs obtain data from Data Banker access programs. The producer and consumers of a particular datum can be written without knowing whether or not the Data Banker stores the value or when a stored value is updated. In most cases, neither the producer nor the consumer need be modified if the updating policy changes.

The Data Banker provides values for all data that report on the internal state of a module or on the state of the aircraft. The Data Banker also reports events involving changes in the values that it supplies. The Data Banker is used as long as consumer and producer are separate modules, even when they are both submodules of a larger module. The Data Banker is not used if consumers require specific members of the sequence of values computed by the producer, or if a produced value is solely a function of the values of input parameters given to the producing program.

The choice among updating policies should be based on the consumers' accuracy requirements, how often consumers require the value, the maximum wait that consumers can accept, how rapidly the value changes, and the cost of producing a new value. This information is part of the specification given to the implementor of the Data Banker.

Specifications of the interface to the Data Banker Module can be found in [6].

B:3.3 FILTER BEHAVIOR MODULE

The Filter Behavior Module contains digital models of physical filters. They can be used by other programs to filter potentially noisy data. The primary secrets of this module are the models used for the estimation of values based on sample values and error estimates. The secondary secrets are the computer algorithms and data structures used to implement those models.

B:3.4 PHYSICAL MODELS MODULE

The software requires estimates of quantities that cannot be measured directly but can be computed from observables using mathematical models. The primary secrets of the Physical Models Module are the models; the secondary secrets are the computer implementations of those models.

The structure of the Physical Models Module is given in section C:3.2. Interface specifications are found in [12].

B:3.5 SOFTWARE UTILITY MODULE

The Software Utility Module contains those utility routines that would otherwise have to be written by more than one other module. The routines include mathematical functions, resource monitors, and programs that signal when all modules have completed their power-up initializations.

Because users of the documentation cannot be expected to know which routines are used by more than one module, the documentation of using modules will always contain references to the Software Utility Module.

The secrets of the module are the data structures and algorithms used to implement the programs.

B:3.6 SYSTEM GENERATION MODULE

The primary secrets of the System Generation Module are decisions that are postponed until system-generation time. These include the values of system generation parameters and the choice among alternative implementations of a module. The secondary secrets of the System Generation Module are the method used to generate a machine-executable form of the code and the representation of the postponed decisions. The programs in this module do not run on the on-board computer; they run on the computer used to generate the code for the on-board system.

The structure of the System Generation Module is given in section C:3.4.

C: THIRD-LEVEL DECOMPOSITION

C:1 HARDWARE-HIDING MODULE

C:1.1 EXTENDED COMPUTER MODULE DECOMPOSITION

C:1.1.1 DATA MODULE

The Data Module implements variables and operators for real numbers, time intervals, and bit strings. The data representations, data addressing, and data manipulation instructions built into the computer hardware are the primary secrets of this module. Specifically, the primary secrets are the representation of numeric objects in terms of hardware data types; the representation of bitstrings; and how to access a bit within a bitstring. The primary secrets also include the representation of times for hardware timers, but the module is unconcerned with how the timers are started or how they measure elapsed real-time. The secondary secrets of this module are how range and resolution requirements determine representation; the procedures for performing numeric operations; the procedures used to perform bitstring operations; and how to compute the memory location of an array element given the array name and the element index.

C:1.1.2 INPUT/OUTPUT MODULE

The Input/Output Module transmits bitstrings to and from peripheral devices without any interpretation of the values. It also contains diagnostic programs to test the I/O hardware. Specifically, the primary secrets are hardware instruction sequences to perform I/O operations; the assignment of information to channels and I/O words; how the success or failure of an I/O operation is determined; how the diagnostic tests are performed; the criteria used to judge results; and the timing characteristics that affect test evaluation. The secondary secrets of this module are the techniques used to prevent simultaneous use of resources; when input operations actually take place; and the values written out for unused discrete output word bits.

C:1.1.3 COMPUTER STATE MODULE

The Computer State Module keeps track of the current state of the Extended Computer, which can be either ‘‘operating’’, ‘‘off’’, or ‘‘failed’’, and signals relevant state changes to user programs. The primary secret is the way that the hardware detects and signals state changes. Specifically, the primary secrets are how the hardware behaves when the power is turned on; the effect that the GO/NO-GO timer has on the state of the machine; how the hardware behaves when it enters malfunction states; how malfunctions in hardware functions are detected and reported; how many actual states the hardware has, and what hardware transitions are possible; what internal malfunctions cause transitions from ‘‘operating’’ to ‘‘failed’’; and what causes transitions to ‘‘off’’ and to ‘‘operating’’. After the EC has been initialized, this module signals the event that starts the initialization for the rest of the software.

C:1.1.4 PARALLELISM CONTROL MODULE

The virtual computer provided by the Extended Computer Module executes a set of processes in parallel. The Parallelism Control Module determines the rate of progress of processes subject to the constraints imposed by synchronization operations and by the timing parameters in the process definitions. The synchronization operations, the scheduler, and diagnostic procedures for testing the interrupt hardware are part of this module.

The number of processors, the mechanism for process switching, how the interrupt hardware is tested, the criteria used to judge test results, and the timing characteristics that affect test evaluation are the primary secrets of this module. The secondary secrets of this module are the data structures and operations required to load a process on a processor; the data structures required to represent processes and keep track of their current state; how the exclusion relation is implemented; the data structures and algorithms required to keep track of semaphore values and of process states; and how synchronization operations are made indivisible.

C:1.1.5 PROGRAM MODULE

The Program Module determines the order of statement execution within a process. It provides an instruction that permits loops and conditional selection among alternative code sections; it also provides subprogram invocation.

The primary secrets of this module are the sequence control mechanisms of the actual computer. Specifically, they are the control structures that exist at the hardware level; the hardware instruction sequences needed to implement the EC control structures; and how control gets transferred to the program and later returned to the user program, including any subroutine linkage conventions such as saving and restoring registers. The secondary secrets are how parameter information is communicated between the invoked and invoking programs and how the order of execution is determined.

C:1.1.6 VIRTUAL MEMORY MODULE (H)

The Virtual Memory Module presents a uniformly addressable virtual memory to the other Extended Computer submodules, allowing them to use virtual addresses for both data and subprograms. It provides diagnostic facilities for testing the memory. The primary secrets of the Virtual Memory Module are the hardware addresses for data and instructions; differences in the way that different areas of memory are addressed are hidden; how the memory tests are performed; the criteria used to judge results; and the timing characteristics that affect test evaluation. The secondary secrets of the module are the policy for allocating real memory to virtual addresses; the programs that translate from virtual address references to real instruction sequences; which parts of memory are checked when a diagnostics program is run; and the algorithm used to check that memory.

This module is invisible to Extended Computer users, except for a single program that allows a user to request a memory test. The Extended Computer offers a ‘‘typed’’ memory instead of a memory consisting of general-purpose words.

C:1.1.7 INTERRUPT HANDLER MODULE (H)

The Interrupt Handler Module is responsible for responding to external interrupts and reporting them; to other programs, an interrupt event looks like any other event signalled by the software. As a result, other programmers are not aware of either interrupts or the Interrupt Handler Module.

The primary secrets of this Module are which devices must be polled and which execute interrupts, the way that the hardware behaves when an interrupt occurs, the mapping between hardware interrupts and events signalled by the software, and the method used to identify the source of the interrupt. The secondary secret is the mechanism for translating hardware interrupts into software events.

C:1.1.8 TIMER MODULE

The Timer Module is responsible for measuring elapsed real-time. Its primary secrets are the hardware’s timer mechanisms and the number of actual hardware timing devices and their characteristics, including range and resolution. Its secondary secrets are the data structures and methods used to keep track of the intervals being measured; the data structures needed to keep track of all the active timers; the algorithms needed to process all of them; and which hardware timing device is assigned to a timing task. The actual data representation of times for hardware timers is not a part of this module.

C:1.2 DEVICE INTERFACE MODULE DECOMPOSITION

C:1.2.1 AIR DATA COMPUTER

The primary secrets of the Air Data Computer are how to obtain raw measurements for barometric altitude, true airspeed, and Mach number; the scale, offset, and format of the input data items; the way in which the built-in test operates and the method used to determine if the test was passed or failed; the device-dependent operations that must be applied to the raw measurements from the device in order to produce correct barometric altitude, true airspeed, and Mach number and the relationship between the ADC and the FLR.

C:1.2.2 ANGLE OF ATTACK SENSOR

The primary secrets of the Angle of Attack Sensor are the method used to read angle of attack; the value encoding of the data words from the devices; the corrections that are applied by this module; and the circumstances under which the AOA is unreliable.

C:1.2.3 AUDIBLE SIGNAL DEVICE

The primary secret of the Audible Signal Device is the value encoding of the data word to the device. The secondary secret of this module is the method used to cause the signal to beep on and off.

C:1.2.4 COMPUTER FAIL DEVICE

The primary secrets of the Computer Fail Device are how to signal computer failure to the pilot and other devices and the value encoding of the data words to those devices.

C:1.2.5 DOPPLER RADAR SET

The primary secrets of the Doppler Radar Set are the data representation within the I/O words (scale and offset); the method used to determine if the ground speed and drift angle data words are reliable; the arrival sequence of the data; and the operations that must be done on the raw measurements from the device if the DRS has passed its built-in test, and what parts of that test operation are implemented in the hardware. The Doppler Radar Set also hides the causes of local mode transitions, transitions from one configuration (defined by the scans and points of the radar and the display of information) to another, which are not a result of software action.

The secondary secret of this module is the rate at which the device is sampled.

C:1.2.6 FLIGHT INFORMATION DISPLAYS

The primary secrets of the Flight Information Displays are how to display an azimuth and an elevation displacement, two points on a circle relative to a fixed reference point, and an unsigned decimal number; the value encoding of the data to the devices; the maximum displayable values on the individual displays; and the actual limits of the ADI elevation and azimuth indicators are hidden until system generation time.

C:1.2.7 FORWARD LOOKING RADAR

The primary secrets of the Forward Looking Radar Module are how to measure slant range distance to a point on the ground; how to display a point on a radar screen; the scale, offset, and format of the FLR I/O data words; the way in which the FLR is pointed at a target; whether there is a mechanically steered antenna, or an electronically directed beam; the particular sequence of operations necessary to point the FLR, the coordinate system transformation that must be done, and the antenna slave command; the way that the software can detect that the pilot has set the FLR mode to Terrain Following; the particular items of information required by the FLR during TF and how they are used; the relationship between the FLR and other devices, such as the ADI and the ADC; the details of the FLR built-in test and the method used to determine if the hardware passes the test or not.

C:1.2.8 HEAD-UP DISPLAY

The primary secrets of the Head-Up Display Module are the particular sequence of operations necessary to enable and position the various HUD symbols; the scale and offset of the data words for numeric displays; the method used to generate the symbols, i.e. which symbols are hardware produced and which are produced by software actions such as superimposing basic symbols; the reason for restrictions on certain symbols, such as why the RNGCUE and LSC are exclusive, why the PUC must flash when on, and why the FPM cannot be turned on unless the vertical velocity and acceleration displays are enabled; and which symbols have hardware controlled blinking.

The secondary secrets of this module are the method used to cause certain symbols to be removed from the display and how the HUD test pattern is generated.

C:1.2.9 INERTIAL MEASUREMENT SET

The primary secrets of the Inertial Measurement Set are how to measure aircraft altitude, heading, and velocity; how to adjust the orientation of the platform axes; the particular sequence of operations necessary to provide torquing/slewing commands to the platform gyros (the users of the abstract interface request a correction in terms of an angle of rotation about a particular axis. The abstract interface produces the correct gyro torquing/slewing commands); the scale and offset of input and output data items; the format of the input and output data words; the particular corrections that are applied in computing the interface output values; the algorithm that produces current platform velocities from the IMS incremental velocities and an initial velocity; the fact that the IMS mode switch input is the same for ''off'' and ''none''; the amount of time taken to perform fine rotations of the platform; how it is determined that the IMS is ready for computer control; and the fact that this IMS is sensitive to the latitude of the aircraft.

C:1.2.10 INPUT-OUTPUT REPRESENTATION MODULE (H)

The Input-Output Representation Module provides a set of programs that convert common data types to data representations used by more than one of the hardware devices. Its primary secrets are those hardware numeric data representations common to more than one device.

C:1.2.11 MASTER FUNCTION SWITCH (H)

The Master Function Switch hides a suite of five switches and reports which is currently depressed. Its primary secret is the encoding of the data and the hardware interconnections and exclusions among the switches.

C:1.2.12 PANEL

The primary secrets of the Panel Module are the value encoding of the data from/to the panel hardware and the operations required to generate the characters on the displays.

C:1.2.13 PROJECTED MAP DISPLAY SET

The primary secrets of the Projected Map Display Set (PMDS) are the actual layout of the maps on the film cassettes, the number of different maps on one cassette, and the scales of the maps; and the operations necessary to cause the PMDS device to move and orient the maps. The secondary secrets of this module are the conversions to map format from longitude and latitude and the way in which the map reference point is stored and used.

C:1.2.14 RADAR ALTIMETER

The primary secrets for the Radar Altimeter Module are how to read the altitude of the aircraft above local ground or water level and the value encoding of the data words from the device. The secondary secrets for the module are the corrections that are applied to the data by this module and how invalid data is identified.

C:1.2.15 SHIPBOARD INERTIAL NAVIGATION SYSTEM

The primary secrets of the Shipboard Inertial Navigation System Module are how to read the position, altitude and velocity of a nearby ship carrying SINS transmission equipment and the representation of the SINS data within the I/O words, including scale, offset, and label information. The secondary secrets of this module are the method used to determine if SINS data in each category is valid and the maximum

age that the data may reach and still be considered valid.

C:1.2.16 SLEW CONTROL

The primary secrets of the Slew Control Module are how to read data from a device indicating a two dimensional displacement from an origin; the value encoding of the data items used by these access functions; and the fact that the slew control is not a ‘‘perfect’’ device - it does not return exactly zero when in the center position.

C:1.2.17 SWITCH BANK

The primary secrets of the Switch Bank Module are how to read the positions of all switches that do not affect other hardware devices; the value encoding of the data items used by these access functions; and the maximum number of settings for the fly-to number selector (before program assembly time). The maximum number of switch positions is known to the program at run time.

C:1.2.18 TACAN

The primary secrets of the TACAN Module are how to read bearing and slant range to a TACAN station; the value encoding of the data from the TACAN system; and the conditions that determine whether the TACAN data are valid.

C:1.2.19 VISUAL INDICATORS

The primary secrets of the Visual Indicators Module are how to cause visible indicators to be on, blinking, or off, and the value encoding of the data words to the devices.

C:1.2.20 WAYPOINT INFORMATION SYSTEM

The primary secrets of the Waypoint Information System Module are how to read received data giving the positions of waypoints; the format of received data; and the details of the waypoint I/O device.

C:1.2.21 WEAPON CHARACTERISTICS

There are two primary secrets of the Weapon Characteristics Module. The first is the way that the various weapon characteristics are obtained for use. For example, for some weapon types, they are held in a table. For others, they are retrieved from modules. Secondly, for some weapon types, the setting of the ‘‘retarded’’ ASCU switch has an effect on the weapon characteristics. Which weapons it affects and the particular characteristics it affects are secrets. The secondary secret of this module is the information and the source of the information required to determine the identity of the weapon type on the currently active weapon station(s).

C:1.2.22 WEAPON RELEASE SYSTEM

The primary secrets of the Weapon Release System are how to cause weapons to be prepared and released; details of the ARP, ASCU, and pilot’s grip stick hardware, such as operations necessary to perform certain functions and the encoding of values; the maximum settings of the ARP interval and quantity switch and the number of stations and weapon types are secrets at program design and write time (the actual values are inserted during the program assembly); the way that the ready and active weapon stations are identified; the station priority scheme.

C:1.2.23 WEIGHT ON GEAR

The primary secrets of the Weight On Gear Module are how to tell if the plane is resting on the landing gear and the value encoding of the data from the device.

C:2 BEHAVIOR-HIDING MODULE

C:2.1 FUNCTION DRIVER MODULE DECOMPOSITION

The following table describes the Function Driver submodules and their secrets.

Section	Function Driver	Secret
C:2.1.1	AIR DATA COMPUTER	Where to obtain the barometric sea-level pressure. What new device has been installed and where.
C:2.1.2	AUDIBLE SIGNAL	When the audible signal should be on, off, or beeping.
C:2.1.3	COMPUTER FAIL SIGNAL	When to signal computer failure.
C:2.1.4	DOPPLAR RADAR	When to start and stop the Doppler Radar.
C:2.1.5	FLIGHT INFORMATION DISPLAY	What information should be displayed and when.
C:2.1.6	FORWARD LOOKING RADAR	What the current FLR mode should be. Where and when to position the cursors in the FLR display. Where to aim the FLR.
C:2.1.7	HEAD-UP DISPLAY	Where the movable HUD symbols should be placed. Whether a HUD symbol should be on, off, or blinking. What information should be displayed on the fixed-position displays.
C:2.1.8	INERTIAL MEASUREMENT SET	Rules determining the scale to be used for the IMS velocity measurements. When to initialize the velocity measurements. How much and when to rotate the IMS for alignment.
C:2.1.9	PANEL	What information should be displayed on panel windows. When the enter light should be turned on.
C:2.1.10	PROJECTED MAP DISPLAY SET	What geographical location should be displayed on the map. How the map should be oriented. Where the map indicators should be positioned.
C:2.1.11	SINS	When to start and stop SINS reception.

C:2.1.12 VISUAL INDICATOR	When the visual indicators should be on, off, or blinking.
C:2.1.13 WEAPON RELEASE	When to prepare and release a weapon.
C:2.1.14 GROUND TEST	When and how to use the EC test outputs.

C:2.2 SHARED SERVICES MODULE DECOMPOSITION

The Shared Services Module comprises the following modules.

C:2.2.1 MODE DETERMINATION MODULE

The Mode Determination Module determines the system modes (as defined in the requirements document) and the local modes (aliases for system modes) as defined for the individual function driver documents. It signals the occurrence of mode transitions and makes the identity of the current modes available. The primary secrets of the Mode Determination Module are the mode transition tables in the requirements document and the local mode definitions: what causes transitions among the Requirements-defined modes. The secondary secrets of this module are representation of the correspondence between defined mode names and the defining modes; algorithm for signalling mode changes; and how the mode transition criteria is represented.

C:2.2.2 PANEL I/O SUPPORT MODULE

The Panel I/O Support Module provides formatting services for the Function Drivers that display and accept data through the panel. The primary secrets are the required data display and input formats. This module will not be the one that hides the hardware/software interface of a particular panel; it will hide characteristics of the virtual panel created by the Device Interface Module, providing a more convenient interface. This module also signals events about panel operations. The secondary secret of this module is how the interface of the DIM virtual panel is used to implement the services provided by the Input submodule.

Specifically, the primary secrets of the submodules of the Panel I/O Support Module are:

PANEL CONFIGURATION

The data number associated with most of the display items.

Which configurations occur simultaneously.

The way the various panel switches affect the panel control configurations; which configurations require keyboard input, and what input is required in such cases.

PANEL DISPLAY FORMAT

How the format lights are used to display certain data formats.

Certain format rules that govern how values are displayed, such as: how signs (positive/negative) are displayed; how bitstrings and booleans are displayed; when decimal points are used, and when they are merely implied.

PANEL INPUT

The sequence of operations necessary to perform panel input;

When and how each panel input operation is performed; how long ago each input operation occurred.

How the pilot enters the identifying integer for multiple-value panel input items; whether he employs the panel or some other means.

How to tell when a panel input operation has started and terminated.

The secondary secrets of this module include how this module makes use of other Panel I/O submodules to tell what value is being updated by any input operation.

C:2.2.3 SHARED SUBROUTINE MODULE

The Shared Subroutine Modules hide parts of the function definitions that are shared, perhaps with some modification, by several functions. Where, in our judgment, the sharing is not a coincidence and a change in one function driver is likely to be accompanied by a similar change in the others, the routines have been included in the Shared Subroutine module to avoid duplication of code and documentation. In some cases, we have made sharing possible by parameterization.

C:2.2.4 STAGE DIRECTOR MODULE

In some modes, the system sequences through stages; in each stage the program is trying to achieve a goal and the end of each stage is marked by the achievement of that goal. Whether or not a goal has been achieved is determined by the program itself, rather than by an external event. Although many of the stages occur in several modes, the modes differ in the definition of the goals and the sequence of stages.

There are stage directors for each of the alignment modes and for the ground test mode. The primary secret of each Stage Director Module is the sequence of stages and the predicates that determine when a stage transition occurs. The behavior required in the individual stages is a secret of Function Driver submodules, but the rules determining when to proceed from one stage to the next are hidden in the Stage Director Module. Specifically, the primary secrets for the Stage Director Module are as follows:

The criteria for determining the current alignment stage.

What causes one stage to end and another to begin.

The order of stages in each of the alignment modes.

The criteria for determining the current test stage.

The order of test stages.

The secondary secrets of this module are how the stage transition criteria are represented and the algorithms for detecting a transition.

C:2.2.5 SYSTEM VALUE MODULE

The System Value submodule computes a set of values. Although some values are used by more than one Function Driver, the module may include a value that is only used in one Function Driver if the rule used to calculate that value is the same as that used to calculate other shared values. The secrets of the System Value submodule are the rules in the requirements that define the values that it computes. The shared rules in the requirements specify such things as 1) selection among several alternative sources, 2) applying filters to values produced by other modules, or 3) imposing limits on a value calculated elsewhere. The System Value submodule is also responsible for signalling events that are defined in terms of the values it computes.

The individual submodules within the System Value Module are as follows:

DEVICE REASONABLENESS

Primary secrets of this module are the criteria to judge validity or reasonableness of sensor inputs and when and how often the reasonableness tests are performed.

IMS ALIGNMENT

Primary secrets of this module are when the navigation/alignment timer is started, stopped, and reset, and when the alignment tests are performed, and the criteria defining success or failure of the test.

REFERENCE POINT

The primary secrets of the module are the definition of the HUD reference point, fly-to point, called-up point, target, offset aim point, adjusted point, and fix point; what actions are taken to designate the reference points; what device determines the fix point and under what conditions; the latitude/longitude error reference points; for ground and slant ranges, which of the available earth terrain models is used for the calculations; how the value of the Mark number is determined; and where the destination and mark locations come from, when they are updated, and with what values.

SLEW

Primary secrets of the module include the rules for when slew operations may be performed; the rules for determining the rates at which slewed symbols or devices move; how the program decides if the current state is before slewing, after slewing, or during slewing; and what constitutes a legal slew input.

VALUE SELECTION

Value Selection's primary secrets are which sensors provide the values produced; how the choice among various sensors is made; how the choice between providing a sensor value or providing a null value is made; and when various values are determined and/or updated.

WEAPON RELEASE

Primary secrets of the module are how the number of weapons in a stik is determined; how the delivery spacing for a stik is determined; what classes of weapons cannot be delivered in a stik; what configurations determine a high or low drag release condition; what weapon types are applicable to the conditions; how to tell if the conditions for a successful weapon release exist; how to tell if a target is in range or not; how to tell what weapon delivery steering stage the system is in (if any); and how to determine the radius for blast avoidance.

C:3 SOFTWARE DECISION MODULE

C:3.1 APPLICATION DATA TYPE MODULE DECOMPOSITION

The Application Data Type Module is divided into two submodules.

C:3.1.1 NUMERIC DATA TYPE MODULE

The Numeric Data Type Module implements the following variables types: accelerations, acceleration vectors, angles, angular rates, densities, displacements, distances, orientations, orientation rates, pressures, speeds, and velocities. These modules may be used to implement types with restricted ranges or special interpretations (e.g., angle is used to represent latitude). Primary secrets of the module are the representation of numeric objects; how range and resolution information is used to determine representation; the procedures for performing numeric operations; and the conversions required if two objects of the same type or type class are not represented in the same way.

C:3.1.2 STATE TRANSITION EVENT DATA TYPE MODULE

This module allows programs to create and operate on data types described as finite state machines. The domain of a variable is a relatively small set of states. A change in the value of a variable or a change in its state is an event that can be signalled and awaited. State changes in such variables are reported to user programs [11]. The module's primary secrets are the algorithms used in the programs corresponding to application data type attributes; the internal representation of states, sets, and relations; and how processes await a state transition event and how they are restarted.

C:3.2 DATA BANKER MODULE

The Data Banker Module comprises the modules described below.

C:3.2.1 SINGULAR VALUES MODULE

This module consists of programs that return values and report events provided by a single producer program. Its secrets are the update policies used for the values, and the detection mechanism used for the events.

C:3.2.2 COMPLEX EVENT MODULE

This module consists of programs that report events that depend upon more than one produced value. Its secret is how the Data Banker detects these events.

C:3.3 FILTER BEHAVIOR MODULE

No third level modules exist for this module.

C:3.4 PHYSICAL MODELS MODULE DECOMPOSITION

The Physical Models Module comprises the modules described below.

C:3.4.1 AIRCRAFT MOTION MODULE

The Aircraft Motion Module hides models of the aircraft's motion which are used to calculate aircraft position, velocity and attitude from observable inputs. Secrets of the module include the methods of converting the aircraft's potential energy to kinetic energy; the ratio(s) of thrust to drag when the aircraft is accelerating with maximum indicated normal acceleration; and basic physics logic, such as the equations of motion and acceleration and the geometry of torques and lever arms.

C:3.4.2 EARTH CHARACTERISTICS MODULE

The Earth Characteristics Module hides models of the earth and its atmosphere. This set of models includes models of local gravity, curvature of the earth, air pressure, magnetic variation, local terrain, rotation of the earth, Coriolis force, and atmospheric density. Secrets of the module also include how the angles to impact point and how the slant range to impact point are determined for air-to-ground launch weapons; how trajectories are determined for air-to-air launch weapons; which, if any, of these calculations must be performed periodically instead of on demand; which weapons characteristics and which values are used in calculation of trajectories and how they are used; and in which reference frame the calculations are implemented.

C:3.4.3 HUMAN FACTORS MODULE

The Human Factors Module is based on models of pilot reaction time and perception of simulated continuous motion. The models determine the update frequency appropriate for symbols on a display.

C:3.4.4 TARGET BEHAVIOR MODULE

The Target Behavior Module contains models used to predict target behavior, such as whether it is stationary or moving.

C:3.4.5 WEAPON BEHAVIOR MODULE

The Weapon Behavior Module contains models used to predict weapon behavior after release.

C:3.5 SOFTWARE UTILITY MODULE

C:3.5.1 POWER-UP INITIALIZATION MODULE

The Power-Up Initialization Module signals when the following modules have completed their power-up initialization procedures: Function Driver, Shared Services, Device Interface, Physical Models, Data Banker, Numerical Algorithms.

C:3.5.2 NUMERICAL ALGORITHMS MODULE

This module provides mathematical service routines needed by more than one module within the system. These functions include services for data manipulations such as exponentiation, square root and logarithm. The module also provides commonly used STE [AT] specific types. The primary secrets of this module are the algorithms implementing the functions.

C:3.6 SYSTEM GENERATION MODULE DECOMPOSITION

C:3.6.1 SYSTEM GENERATION PARAMETER MODULE

The System Generation Parameter Module provides values for all the system generation parameters defined in other modules, including those specified in module interfaces and those defined in the module implementations. There is a submodule of the System Generation Parameter Module for each module in the rest of the system; each of these submodules is in turn composed of an external parameter submodule and an internal parameter submodule. External parameters of a module are available to other modules; internal parameters are secrets of the module. The primary secrets of this module are the values of the parameters for a particular version of the system.

C:3.6.2 SUPPORT SOFTWARE MODULE

The support software in this system is a combination of commercially available tools such as operating systems, compilers, editors, and revision control aids, plus a small set of tools built especially for this project. A report describing the tools will be made available upon termination of the project.

Acknowledgements

P. C. Clements' efforts have led to substantial improvements in the structure described in this document and to substantial clarifications in the writing. Comments on earlier drafts by B. Amlicke, S. Bouchard, L. Chmura, H. Elovitz, C. Heitmeyer, R. Krutar, D. Weiss, and J. Shore were very helpful. Thanks to Cheryl Hinson for converting the format and layout of the document text to its current form.

IV. REFERENCES

[1] Heninger, K., Kallander, J., Parnas, D., and Shore, J.; *Software Requirements for the A-7E Aircraft*; NRL Memorandum Report 3876; 27 November 1978.

[2] Parnas, D.; ''On the Criteria To Be Used in Decomposing Systems into Modules''; *Comm. ACM*, Vol. 15, No. 12 (December 1972), pp. 1053-1058.

[3] Parker, A., Heninger, K., Parnas, D., and Shore, J.; *Abstract Interface Specifications for the A-7E Device Interface Module*; NRL Memorandum Report 4385, 20 November 1980.

[4] Heninger, K., Clements, P., Parnas, D., and Weiss, D.; *Interface Specifications for the A-7E (SCR) Extended Computer Module*; NRL Memorandum Report 5502, 31 December 1984.

[5] Clements, P.; *Interface Specifications for the A-7E Shared Services Module*; NRL Memorandum Report 4863, 8 September 1982.

[6] Clements, P., and Faulk, S.; *Interface Specifications for the SCR (A-7E) Data Banker Module*; NRL Technical Memorandum 7595-020, 30 June 1986.

[7] Parnas, D.; ''Designing Software For Extension and Contraction'', *Proceedings of the 3rd International Conference on Software Engineering* (10-12 May 1978), pp. 264-277.

[8] Parnas, D., and Wuerges, H.; ''Response to Undesired Events in Software Systems''; *Proc. Second Int. Conf. Software Eng.*, pp. 437-446; 1976.

[9] Clements, P.; *Function Specifications for the A-7E Function Driver Module*; NRL Memorandum Report 4658, October 1981.

[10] Clements, P., Faulk, S., and Parnas, D.; *Interface Specification for the A-7E Applications Data Type Module*; NRL Memorandum Report 8734, 23 August 1983.

[11] Clements, P.; NRL Report 8734, 23 August 1983.

[12] Labaw, B.; *Abstract Interface Specifications for the A-7E Physical Models Module*; NRL Technical Memorandum 7595-021, 4 September 1986.

[13] Labaw, B.; *Interface Specifications for the A-7E Filter Behavior Module*; NRL Technical Memorandum 7595-022, 20 August 1986.

[14] Clements, P., Labaw, B., and Wicinski, T.; *Interface Specifications for the A-7E Software Utilities Module*; NRL Technical Memorandum 7595-023, 23 June 1986.

[15] Parnas, D.; Software Engineering Principles, University of Victoria, 1982.

[16] Parnas, D., Clements, P.; "A Rational Design Process: How and Why to Fake It"; *IEEE Transactions on Software Engineering*; Vol. SE-12, No. 12, (February 1986), pp. 251-257.

V. GLOSSARY

abstract interface — an abstraction that represents more than one interface (see **interface**, **module interface**); it consists of the assumptions that are included in all of the interfaces that it represents.

abstraction — a description of a set of objects that applies equally well to any one of them. Each object is an instance of the abstraction.

access function — see **access program**.

access program — a program that may be called by programs outside of the module to which it belongs. Most run-time communication between modules is effected by invocation of access programs. There are several different sorts of access functions: some return information to the caller, some change the state of the module to which they belong, and some do both.

consumer — a program that requires data produced by another program.

event — (1) change in a condition; (2) signal from a module to its user programs indicating the occurrence of some change within the module. Events resemble hardware interrupts because they occur at unpredictable times and are not synchronized with the control flow of user programs. In the A-7E program, events will be signalled to user programs using an event mechanism (see below).

event mechanism — a programming construct that allows processes to communicate about the occurrence of events. Typical operations provided include an operation to wait for a particular event to occur and an operation to signal that a particular event has occurred.

hidden submodule — a submodule whose existence is part of the secret of the parent module.

interface — (1) between two programs: the assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his own program.

(2) between a program and a device: assumptions about the device that must be accounted for in the program in order for the program to work as expected.

internal program — a program that is not accessible to programs outside the module; the existence of the internal program is part of the secret of the module.

module — a programming work assignment consisting of one or more programs. A module may be divided into smaller modules (submodules).

module facility — the access programs and events provided by a module in order to allow user programs to be independent of the module secret. A complete description of a facility is a specification of the module.

module hierarchy — a hierarchy defined by the relation ‘‘contains’’ on pairs of modules.

module implementation — the algorithms, data structures, and programs that satisfy the module specification.

module interface	the set of assumptions that the authors of external programs may make about the module. It includes restrictions on the way that the module may be used. In the A-7E software, modules communicate either by one module using access programs from the other module, or by one module being notified of an event that was signalled by the other module. The interface consists of assumptions about the availability of the access programs, the syntax of the calls on the access programs, the behavior of the access programs, and the meaning of events. See also **interface**.
module secret	see **secret**, **module implementation**.
module specification	a description of a module interface; see also **module facility**.
module's structure	the way that a software module is divided into submodules and programs.
primary secret	the characteristics other than decisions by the module designer that a module is intended to hide. See also **secondary secret**.
process	a subset of the run-time events of the system used as administrative units in the run-time allocation of processors. See also **process definition**.
process definition	the program that controls the sequence of actions by a process.
producer	a module that provides data for use by other programs. A program that returns to its caller an output parameter that is a function of the input that the caller provided is not considered a data producer. Examples of data producers include programs that read values measured by sensors, or calculate physical characteristics based on mathematical models.
program	a named, machine executable, description of an algorithm. The name may be used to invoke the program's execution. A program may include a description (declaration) of the data structures that it uses; it may invoke other programs and refer to data structures that have been described in other programs. See also **subprogram**, **process definition**.
required function	used in the sense of [1]. Each requirements function is the determination of the value of a specific, closely related set of output values. The system performs all of its requirements functions when it properly determines the value of all of its outputs.
secondary secret	software design decisions made to implement the abstraction that hides the primary secret.
secret	the facts about the module that are **not** included in its interface; i.e., assumptions that user programs are not allowed to make about the module. The correctness of programs in other modules must not depend on those facts. The secrets tell how the module's specification has been satisfied. See also **module specification**, **primary secret**, **secondary secret**.
submodule	any module that is a component of a higher level module.
subprogram	a subprogram is a program that can be invoked by another program. A subprogram may be either a subroutine or a macro.
sysgen parameters	a symbol used as a placeholder for values that will be supplied just before a system is generated.

undesired event (UE) — a run-time event that the designers hope will not occur. Production versions of the A-7E program are written on the assumption that they do not occur.

undesired event assumption — assumptions about what constitutes improper use of a module by user programs, e.g., calling an access program with parameters of the wrong type.

use, uses — Program A uses program B if there must be a correct version of B present for A to run correctly. A program uses a module if it uses at least one program from that module. A module uses another module if at least one program uses that module.

user programs — all programs that use programs from a module but are not part of that module. The term ‘‘user’’ is relative to the module being discussed.

virtual computer — a computer-like set of instructions implemented, at least in part, by software.

virtual machine — see **virtual computer**.

visible submodules — submodules whose existence is visible to user programs.

TABLE OF CONTENTS

Use the footer of this page